Group Project 3: Distribution and Scalability

CECS 327 – Intro to Networks and Distributed Computing

You should submit the required materials on Canvas by **11:55pm, May 3rd (Saturday), 2025.**

Overview

This project extends the previous peer-to-peer (P2P) network via Docker containers. You will enhance your P2P network to support decentralized storage and retrieval. Each node will be able to:

- Upload and download files
- Insert and query key-value pairs
- Distribute storage responsibilities using a basic Distributed Hash Table (DHT)

This phase introduces data layer concepts essential for scalable, decentralized systems.

Project Phases & Steps (Provided codes are just examples and free to change)

Phase 1: File Upload & Download

Goal:

Allow each node to store and serve files from a shared volume.

Instructions:

1. Create a local storage/ directory in each node's container.
2. Add two Flask endpoints to handle file uploads and downloads

```
%That's a sample code, you can change it
@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    file.save(f"./storage/{file.filename}")
    return jsonify({"status": "uploaded", "filename":
file.filename})

@app.route('/download/<filename>', methods=['GET'])
def download_file(filename):
    return send_from_directory('./storage', filename)
```

3. Mount a volume in Docker so the files persist:

```
docker run -d -p 5001:5000 -v "$(pwd)/storage:/app/storage" --name
node1 p2p-node
```

4. Test file upload:

```
curl -F 'file=@mydoc.txt' http://localhost:5001/upload
```

5. Test file download:

   Visit http://localhost:5001/download/mydoc.txt in your browser.

Phase 2: Key-Value Store

Goal:

Add basic data storage capability to each node for storing and querying key-value pairs.

Instructions:

1. Maintain an in-memory dictionary
2. Add Flask key-value endpoints
3. Test storing and retrieving values:

```
curl -X POST http://localhost:5001/kv -H "Content-Type:
application/json" -d '{"key": "color", "value": "blue"}'
```

```
curl http://localhost:5001/kv/color
```

Phase 3: Add DHT-Based Routing for Storage

Goal:

Distribute storage responsibility using hashing.

Instructions:

1. Implement SHA-1 hashing
2. On each `/kv POST` or `/kv/<key> GET`:
    - Use `hash_key_to_node` to find the node responsible.
    - If the current node **is not responsible**, **forward** the request using requests.post() or requests.get().

    Example forwarding:

```
if current_node != responsible_node:
    res = requests.post(f"{responsible_node}/kv", json=data)
```

Project Options (Bonus points are given)

Option 1:

Peer Health Monitoring & Fault Tolerance

Goal: Introduce reliability features and detect node failures.

Activities:

- Implement periodic **heartbeat messages** among peers.
- Use a thread or asyncio to check for unresponsive peers.
- Update peer list dynamically by removing inactive peers.

Option 2:

Visualization and Monitoring

Goal: Provide visibility into the network's behavior.

Activities:

- Use **Flask + JavaScript (D3.js or Chart.js)** to create a dashboard showing:
  - Active peers
  - Network graph (nodes and connections)
  - Message traffic over time
- Add logs or Prometheus-style metrics for debugging.

Option 3:

Add a Gossip Protocol

Goal: Use decentralized dissemination of peer information.

Activities:

- Periodically share peer lists with random neighbors.
- Limit message flooding using TTL (time-to-live) metadata.

---

3. Required Deliverables

1. README File: Instructions on how to build, run, and test your system.
2. Source Code: Include a Makefile (if applicable) and ensure the submission is in the correct format.
3. Project Report (PDF):
   - Explanation of the system design.
   - Screenshots of your design and outputs.
4. Execution Demonstration Video:
   - Record a video showing your code execution and outputs.
   - The video should display your name and date as identification.
   - Upload to **YouTube** (or another platform) and provide a link in your report.

4. Submission Guidelines

- Submit a single .zip/.rar file containing all required files (zip named by your Name).
- Only one submission per group is required.
- Ensure your code compiles and runs; otherwise, a zero grade will be assigned.
- If your code is incomplete, specify missing parts in your report for partial credit consideration.
- Provide sufficient comments in your code to explain the logic.

5. Grading Criteria

Details	Points
Have a README file shows how to compile and test your submission	5 pts
Submitted code has proper comments to show the design	15 pts
Screen a ***video*** to record code execution and outputs	15 pts

Have a **report** (pdf or word) file explains the details of your entire design	20 pts
Report contains clearly individual contributions of your groupmates	5 pts
Code can be compiled and shows correct outputs	40 pts

6. Policies

1. Late Submissions: Will be penalized as per course syllabus.
2. Plagiarism: Code-level discussions are prohibited. Anti-plagiarism tools will be used.
3. Use of AI Tools: ChatGPT, GPT-4, and similar AI tools are prohibited for both code and written content.

Final Notes:

- This project requires independent research and problem-solving skills.
- Properly cite any resources you reference.
- Have fun experimenting with distributed systems and networking!

Good luck! 🚀